1. ○

CS305-2023Fall 项目报告

引言

大家好，

今天，我们将向大家介绍我们的CS305-2023Fall项目，该项目的重点是基于HTTP/1.1协议实现一个文件管理服务器。我们的项目旨在创建一个强大而高效的服务器，使客户端能够执行各种文件管理操作，例如查看、下载、上传和删除文件。同时我们为了方便测试以及用户的体验，我们还写了一个html文档以供方便展示。

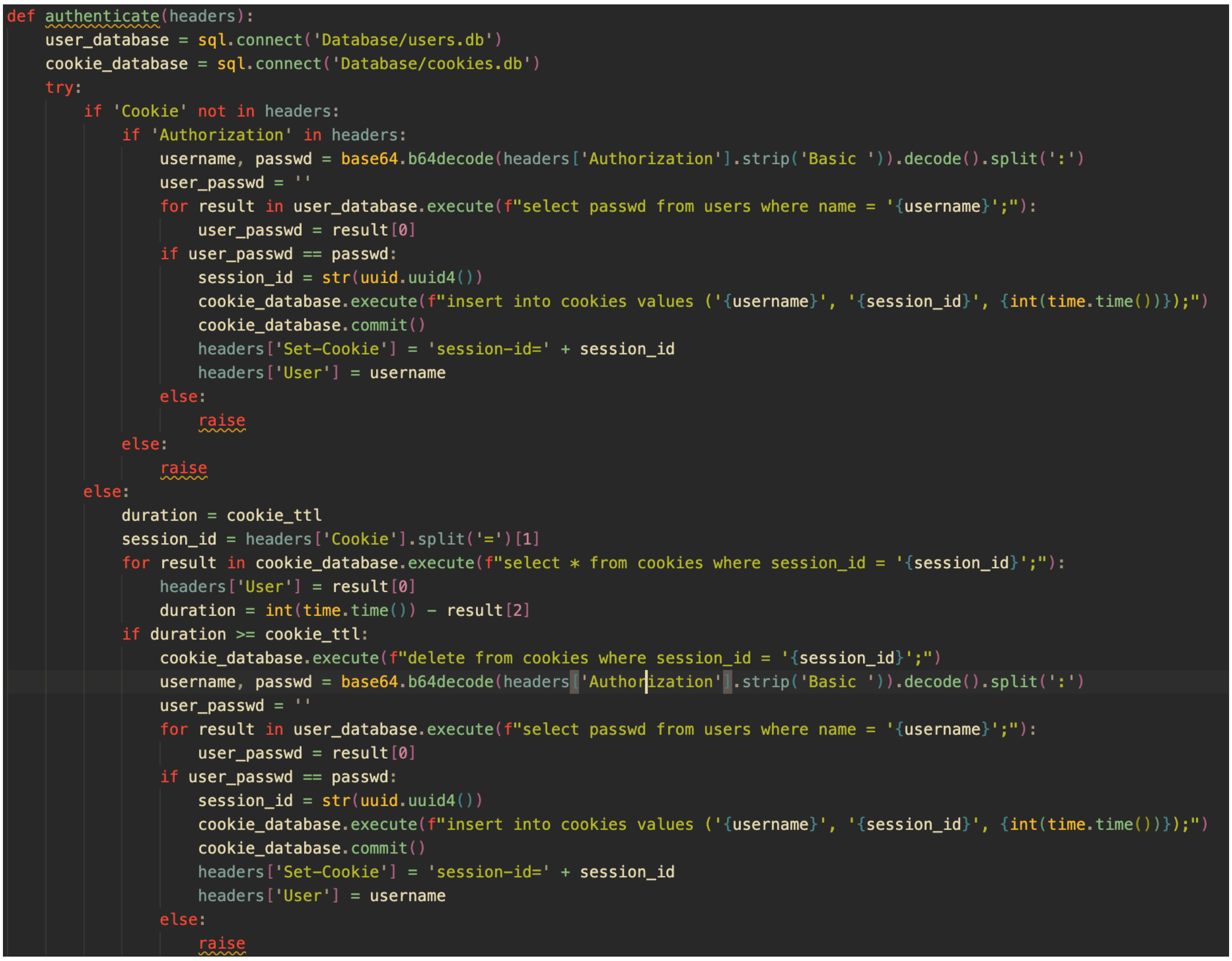
框架设计

为了实现我们的目标，我们设计了一个自定义的HTTP服务器框架，并且自主实现了Python中的Socket API（TCPServer.py）。该框架提供了处理传入的HTTP请求、解析请求和生成适当响应的必要功能。它使我们能够构建一个可扩展和可靠的文件管理服务器。

信息认证和Cookies

代码使用SQLite数据库连接，分别连接了`users.db`和`cookies.db`两个数据库文件。这些数据库用于存储用户信息和会话信息。

- 验证 Cookie：函数首先检查请求头部中是否包含`Cookie`字段。如果存在，它将提取出会话 ID，并在`cookies.db`数据库中查找相应的会话信息。
- 用户名密码验证：如果未提供 Cookie，函数将检查请求头部中是否包含`Authorization`字段。如果存在，它将提取出用户名和密码，并在`users.db`数据库中查找相应的用户信息。
- 会话创建和更新：如果用户名和密码验证成功，函数将生成一个新的会话 ID，并将会话信息插入到`cookies.db`数据库中。会话 ID 通过设置`Set-Cookie`响应头部返回给客户端，同时对于一个`cookies`,设置他的有效时间为一小时，如果时间超出需要重新返回认证。
- 异常处理：如果验证失败或发生异常，函数将设置`User`字段为`None`，并在响应头部中添加`WWW-Authenticate`字段，要求进行基本身份验证。

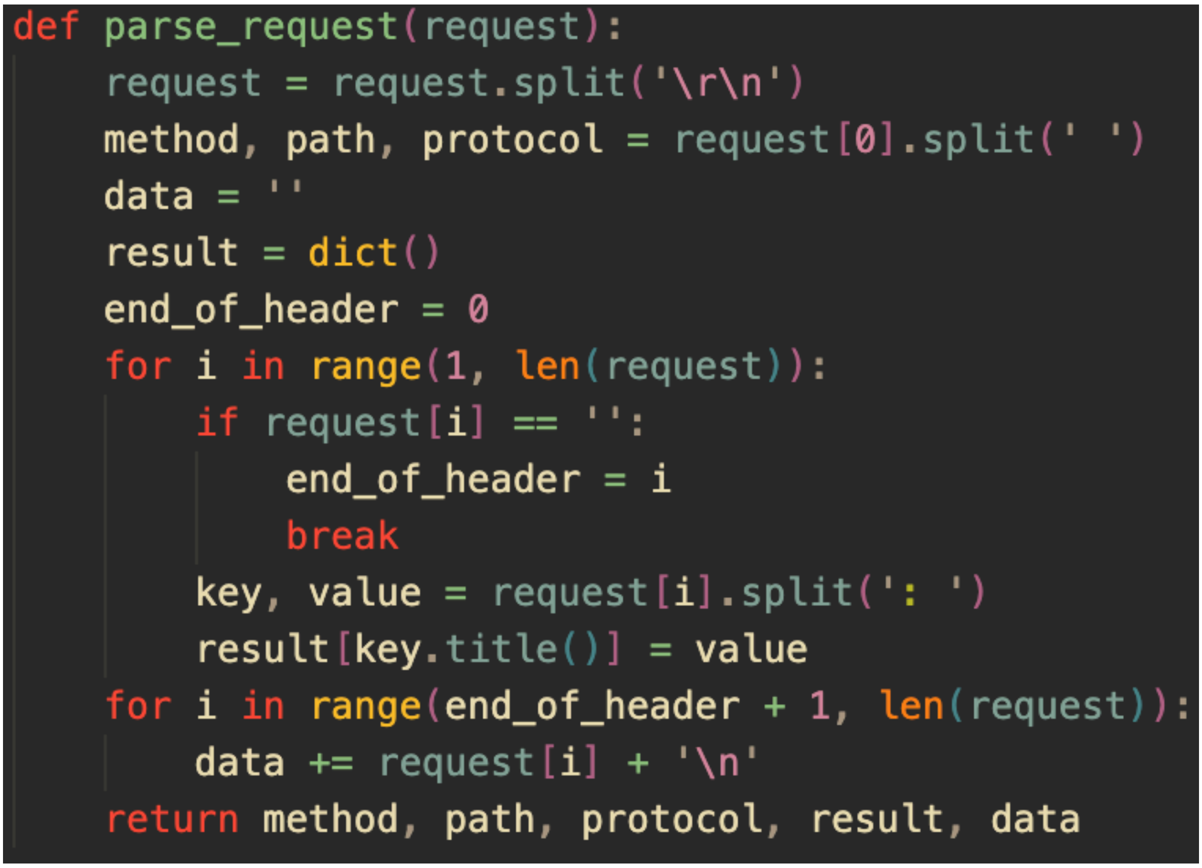


```python
def authenticate(headers):
    user_database = sql.connect('Database/users.db')
    cookie_database = sql.connect('Database/cookies.db')
    try:
        if 'Cookie' not in headers:
            if 'Authorization' in headers:
                username, passwd = base64.b64decode(headers['Authorization'].strip('Basic ')).decode().split(':')
                user_passwd = ''
                for result in user_database.execute(f"select passwd from users where name = '{username}';"):
                    user_passwd = result[0]
                if user_passwd == passwd:
                    session_id = str(uuid.uuid4())
                    cookie_database.execute(f"insert into cookies values ('{username}', '{session_id}', {int(time.time())});")
                    cookie_database.commit()
                    headers['Set-Cookie'] = 'session-id=' + session_id
                    headers['User'] = username
                else:
                    raise
            else:
                raise
        else:
            duration = cookie_ttl
            session_id = headers['Cookie'].split('=')[1]
            for result in cookie_database.execute(f"select * from cookies where session_id = '{session_id}';"):
                headers['User'] = result[0]
                duration = int(time.time()) - result[2]
            if duration >= cookie_ttl:
                cookie_database.execute(f"delete from cookies where session_id = '{session_id}';")
                username, passwd = base64.b64decode(headers['Authorization'].strip('Basic ')).decode().split(':')
                user_passwd = ''
                for result in user_database.execute(f"select passwd from users where name = '{username}';"):
                    user_passwd = result[0]
                if user_passwd == passwd:
                    session_id = str(uuid.uuid4())
                    cookie_database.execute(f"insert into cookies values ('{username}', '{session_id}', {int(time.time())});")
                    cookie_database.commit()
                    headers['Set-Cookie'] = 'session-id=' + session_id
                    headers['User'] = username
                else:
                    raise
```

处理和表达HTTP消息

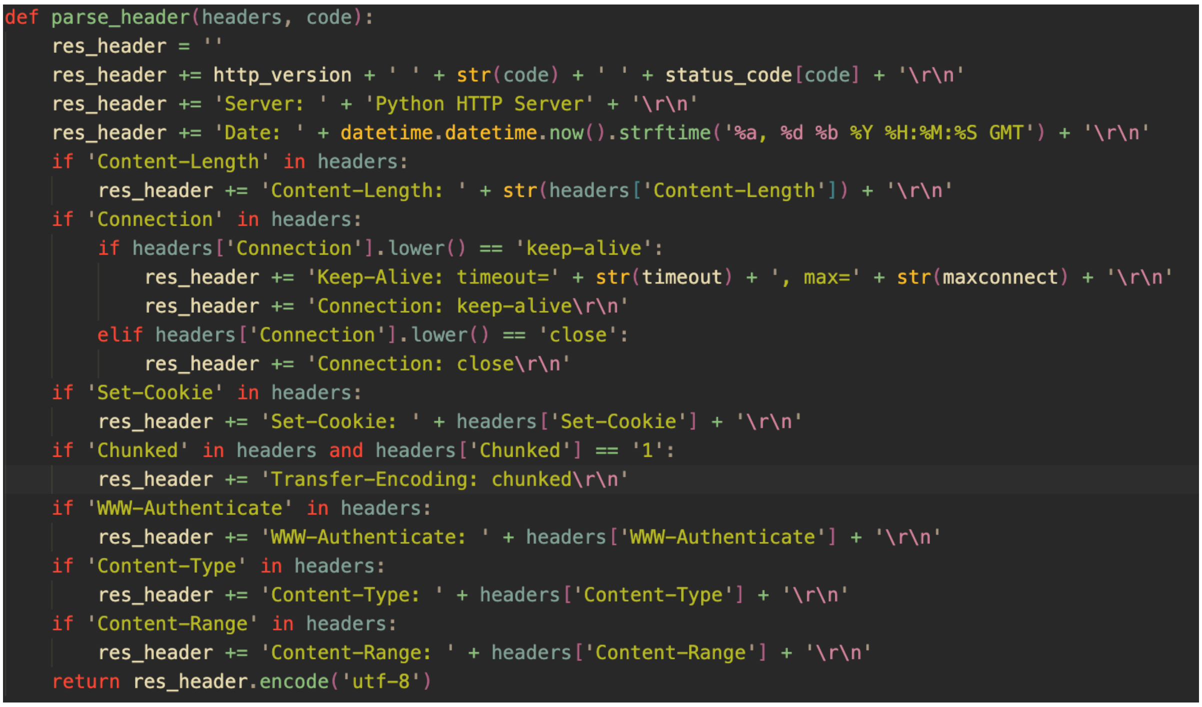
在我们的框架中，我们使用了几种数据结构来处理和表达HTTP消息的每个部分。主要的数据结构包括：

1. 请求（Request）：
   - 方法（Method）：表示请求中使用的HTTP方法（例如GET、POST、DELETE）。
   - URL：存储请求的URL。
   - 头部（Headers）：包含请求中的各种头部信息。
   - 主体（Body）：存储请求体的内容（如果有）。



```python
def parse_request(request):
    request = request.split('\r\n')
    method, path, protocol = request[0].split(' ')
    data = ''
    result = dict()
    end_of_header = 0
    for i in range(1, len(request)):
        if request[i] == '':
            end_of_header = i
            break
        key, value = request[i].split(': ')
        result[key.title()] = value
    for i in range(end_of_header + 1, len(request)):
        data += request[i] + '\n'
    return method, path, protocol, result, data
```

2. 响应（Response）：
   - 状态码（Status code）：指示响应的状态（例如200 OK、404 Not Found）。
   - 头部：存储要包含在响应中的头部信息。
   - 主体：包含响应的内容。

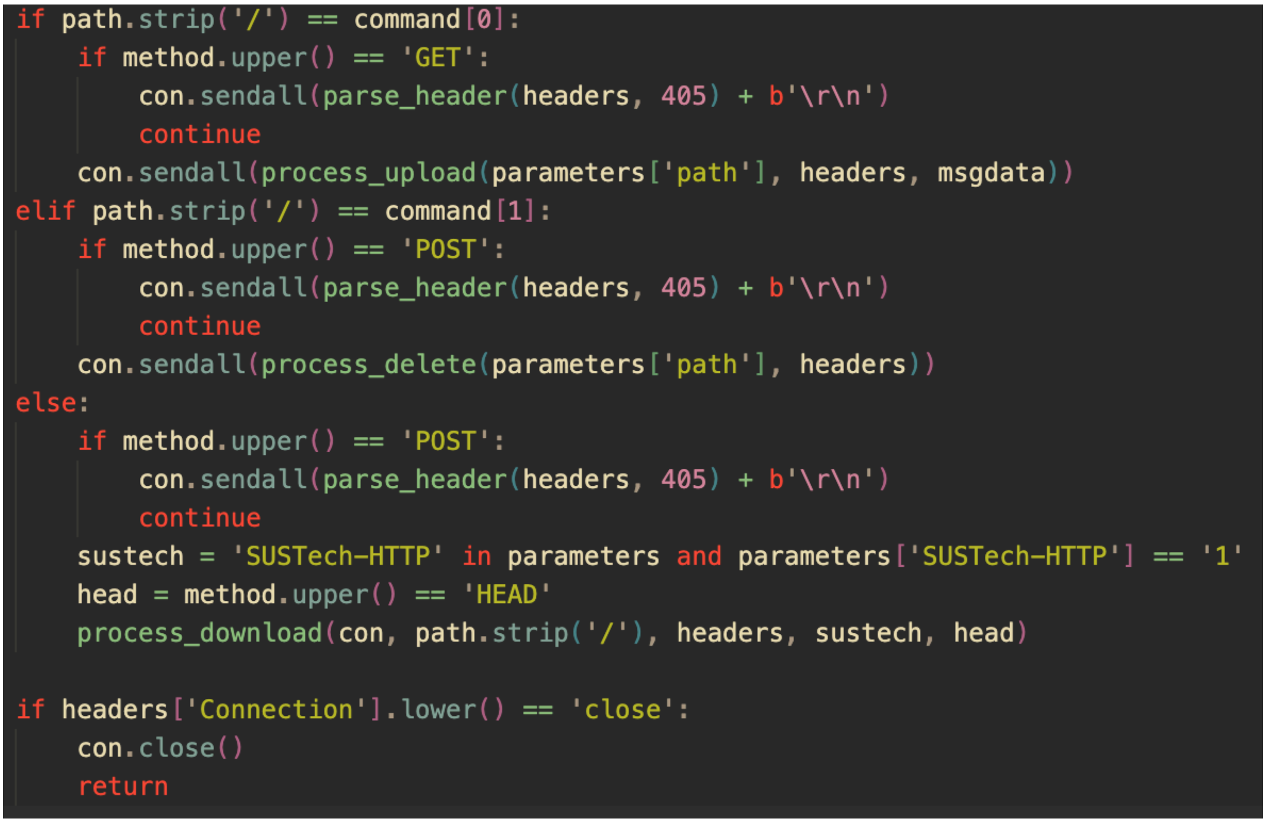


```python
def parse_header(headers, code):
    res_header = ''
    res_header += http_version + ' ' + str(code) + ' ' + status_code[code] + '\r\n'
    res_header += 'Server: ' + 'Python HTTP Server' + '\r\n'
    res_header += 'Date: ' + datetime.datetime.now().strftime('%a, %d %b %Y %H:%M:%S GMT') + '\r\n'
    if 'Content-Length' in headers:
        res_header += 'Content-Length: ' + str(headers['Content-Length']) + '\r\n'
    if 'Connection' in headers:
        if headers['Connection'].lower() == 'keep-alive':
            res_header += 'Keep-Alive: timeout=' + str(timeout) + ', max=' + str(maxconnect) + '\r\n'
            res_header += 'Connection: keep-alive\r\n'
        elif headers['Connection'].lower() == 'close':
            res_header += 'Connection: close\r\n'
    if 'Set-Cookie' in headers:
        res_header += 'Set-Cookie: ' + headers['Set-Cookie'] + '\r\n'
    if 'Chunked' in headers and headers['Chunked'] == '1':
        res_header += 'Transfer-Encoding: chunked\r\n'
    if 'WWW-Authenticate' in headers:
        res_header += 'WWW-Authenticate: ' + headers['WWW-Authenticate'] + '\r\n'
    if 'Content-Type' in headers:
        res_header += 'Content-Type: ' + headers['Content-Type'] + '\r\n'
    if 'Content-Range' in headers:
        res_header += 'Content-Range: ' + headers['Content-Range'] + '\r\n'
    return res_header.encode('utf-8')
```

处理接收到的请求

当服务器接收到一个请求时，我们按照以下流程来处理它：

1. 解析：我们解析接收到的请求，提取相关信息，例如请求方法、URL、头部和主体。
2. 映射：我们将请求目标（URL）映射到服务器中相应的函数。这使我们能够确定所请求资源的适当操作。
3. 执行：映射完成后，我们执行相应的服务器函数来处理请求。这可能涉及任务，如提供目录列表、处理文件下载、处理文件上传或删除文件。
4. 生成响应：在执行服务器函数之后，我们根据请求的结果生成响应。响应包括适当的状态码、头部和响应主体（如果需要）。



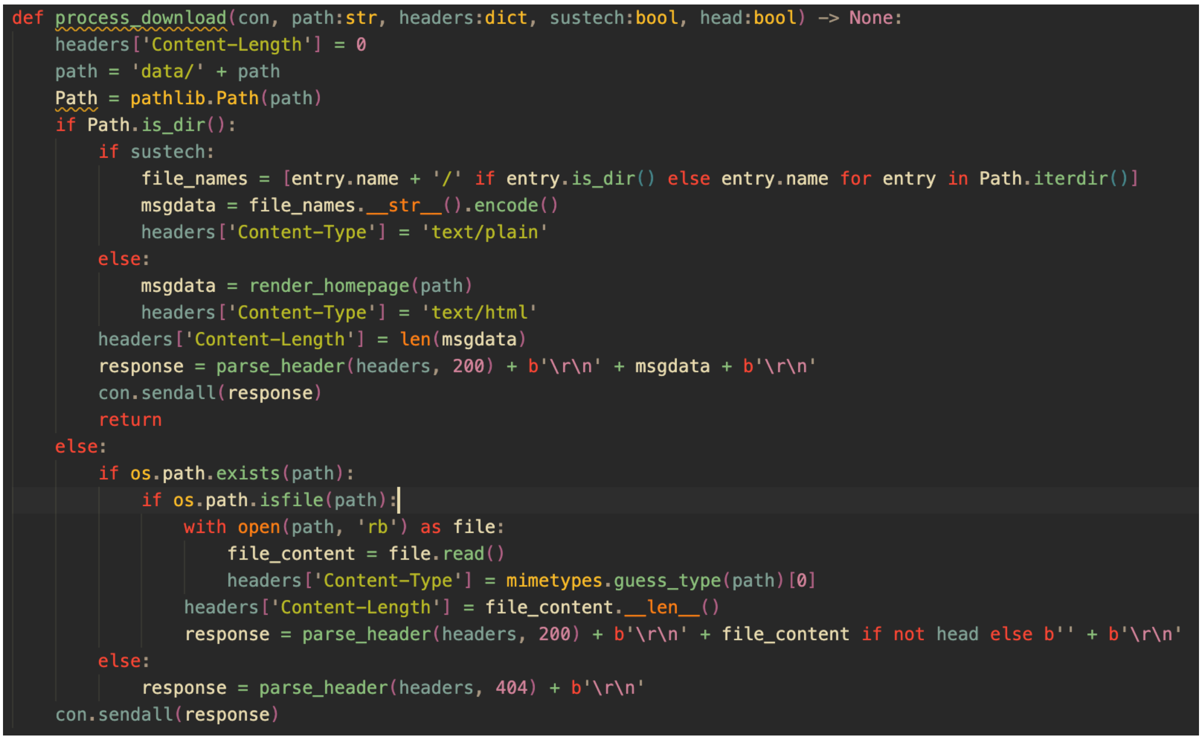
```python
if path.strip('/') == command[0]:
    if method.upper() == 'GET':
        con.sendall(parse_header(headers, 405) + b'\r\n')
        continue
    con.sendall(process_upload(parameters['path'], headers, msgdata))
elif path.strip('/') == command[1]:
    if method.upper() == 'POST':
        con.sendall(parse_header(headers, 405) + b'\r\n')
        continue
    con.sendall(process_delete(parameters['path'], headers))
else:
    if method.upper() == 'POST':
        con.sendall(parse_header(headers, 405) + b'\r\n')
        continue
    sustech = 'SUSTech-HTTP' in parameters and parameters['SUSTech-HTTP'] == '1'
    head = method.upper() == 'HEAD'
    process_download(con, path.strip('/'), headers, sustech, head)

if headers['Connection'].lower() == 'close':
    con.close()
    return
```

基本部分的实现

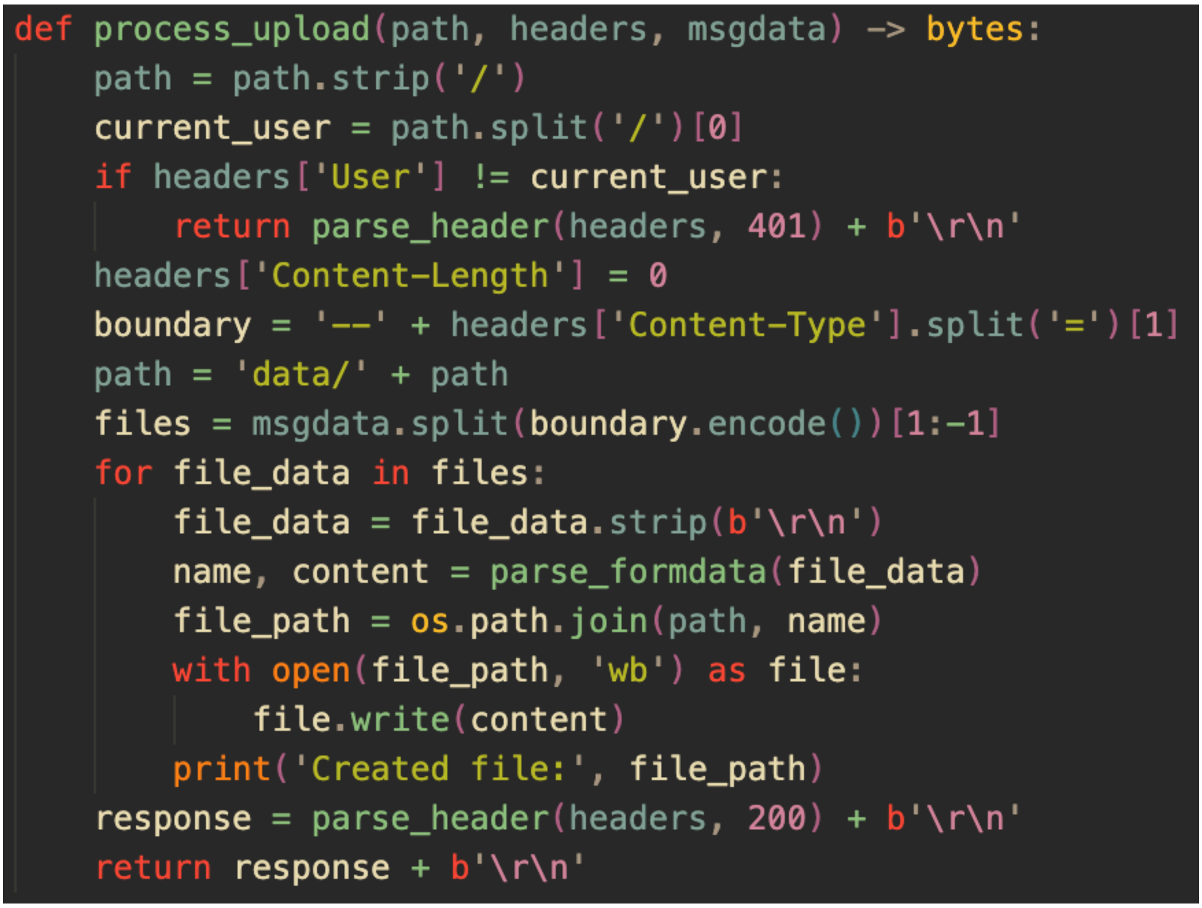
在我们的项目的基本部分中，我们成功实现了以下组件：

1. 基本HTTP服务器：我们使用Python的Socket API构建了一个HTTP服务器，用于处理传入的请求。服务器能够按照HTTP/1.1协议解析和生成响应。
2. 目录列表：我们的服务器提供目录列表功能，允许客户端查看目录的内容。当客户端向目录发送GET请求时，服务器会生成一个HTML响应，列出所有的文件和子目录。
3. 文件下载：客户端可以通过向文件的URL发送GET请求来从服务器下载文件。服务器会读取请求的文件，并将其内容作为响应主体返回。



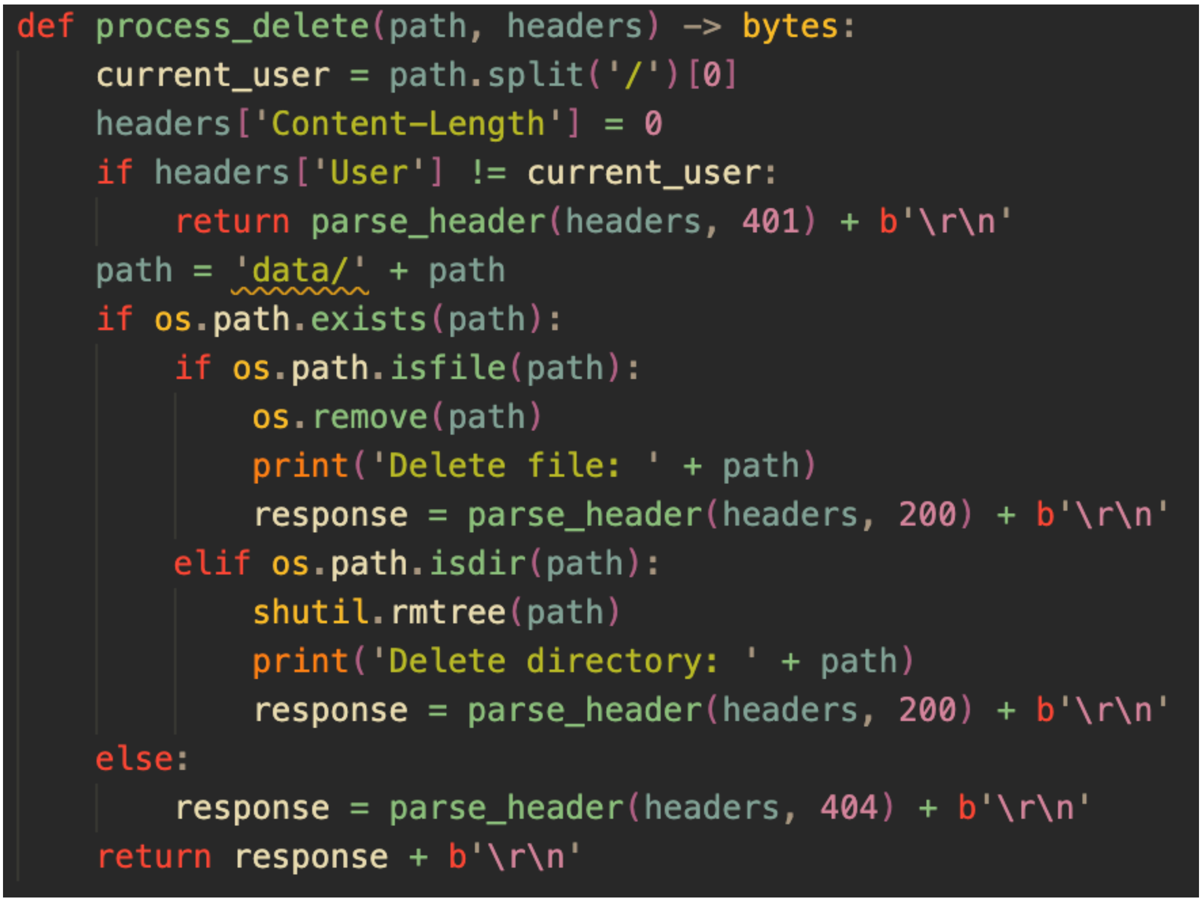
```
def process_download(con, path:str, headers:dict, sustech:bool, head:bool) -> None:
    headers['Content-Length'] = 0
    path = 'data/' + path
    Path = pathlib.Path(path)
    if Path.is_dir():
        if sustech:
            file_names = [entry.name + '/' if entry.is_dir() else entry.name for entry in Path.iterdir()]
            msgdata = file_names.__str__().encode()
            headers['Content-Type'] = 'text/plain'
        else:
            msgdata = render_homepage(path)
            headers['Content-Type'] = 'text/html'
        headers['Content-Length'] = len(msgdata)
        response = parse_header(headers, 200) + b'\r\n' + msgdata + b'\r\n'
        con.sendall(response)
        return
    else:
        if os.path.exists(path):
            if os.path.isfile(path):
                with open(path, 'rb') as file:
                    file_content = file.read()
                    headers['Content-Type'] = mimetypes.guess_type(path)[0]
                headers['Content-Length'] = file_content.__len__()
                response = parse_header(headers, 200) + b'\r\n' + file_content if not head else b'' + b'\r\n'
        else:
            response = parse_header(headers, 404) + b'\r\n'
    con.sendall(response)
```

4. 文件上传：客户端可以通过发送带有文件作为请求主体的POST请求将文件上传到服务器。服务器会处理文件上传，将文件保存到适当的位置，并返回指示上传过程成功或失败的响应。



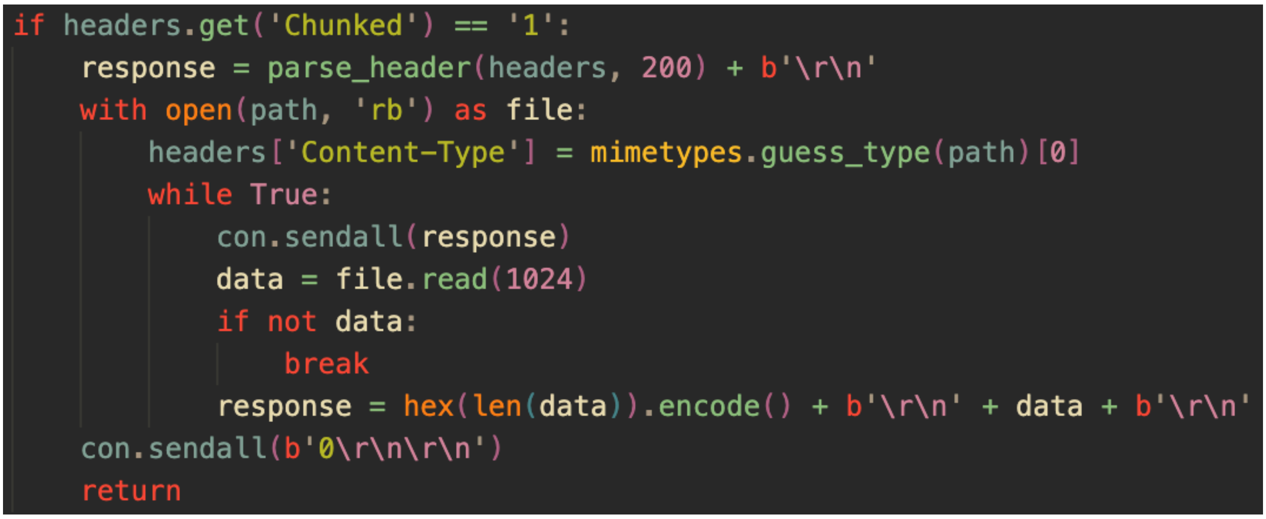
```
def process_upload(path, headers, msgdata) -> bytes:
    path = path.strip('/')
    current_user = path.split('/')[0]
    if headers['User'] != current_user:
        return parse_header(headers, 401) + b'\r\n'
    headers['Content-Length'] = 0
    boundary = '--' + headers['Content-Type'].split('=')[1]
    path = 'data/' + path
    files = msgdata.split(boundary.encode())[1:-1]
    for file_data in files:
        file_data = file_data.strip(b'\r\n')
        name, content = parse_formdata(file_data)
        file_path = os.path.join(path, name)
        with open(file_path, 'wb') as file:
            file.write(content)
        print('Created file:', file_path)
    response = parse_header(headers, 200) + b'\r\n'
    return response + b'\r\n'
```

5. 文件删除：客户端可以通过向文件的URL发送DELETE请求来从服务器删除文件。服务器会处理删除过程，从服务器的存储中删除请求的文件，并返回指示删除过程成功或失败的响应。



```python
def process_delete(path, headers) -> bytes:
    current_user = path.split('/')[0]
    headers['Content-Length'] = 0
    if headers['User'] != current_user:
        return parse_header(headers, 401) + b'\r\n'
    path = 'data/' + path
    if os.path.exists(path):
        if os.path.isfile(path):
            os.remove(path)
            print('Delete file: ' + path)
            response = parse_header(headers, 200) + b'\r\n'
        elif os.path.isdir(path):
            shutil.rmtree(path)
            print('Delete directory: ' + path)
            response = parse_header(headers, 200) + b'\r\n'
    else:
        response = parse_header(headers, 404) + b'\r\n'
    return response + b'\r\n'
```

Chunked Transfer



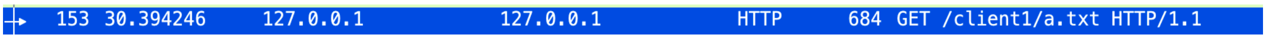
```python
if headers.get('Chunked') == '1':
    response = parse_header(headers, 200) + b'\r\n'
    with open(path, 'rb') as file:
        headers['Content-Type'] = mimetypes.guess_type(path)[0]
        while True:
            con.sendall(response)
            data = file.read(1024)
            if not data:
                break
            response = hex(len(data)).encode() + b'\r\n' + data + b'\r\n'
    con.sendall(b'0\r\n\r\n')
    return
```

按照1024字节为一个单位，然后配上16进制表示的长度和封装发送。

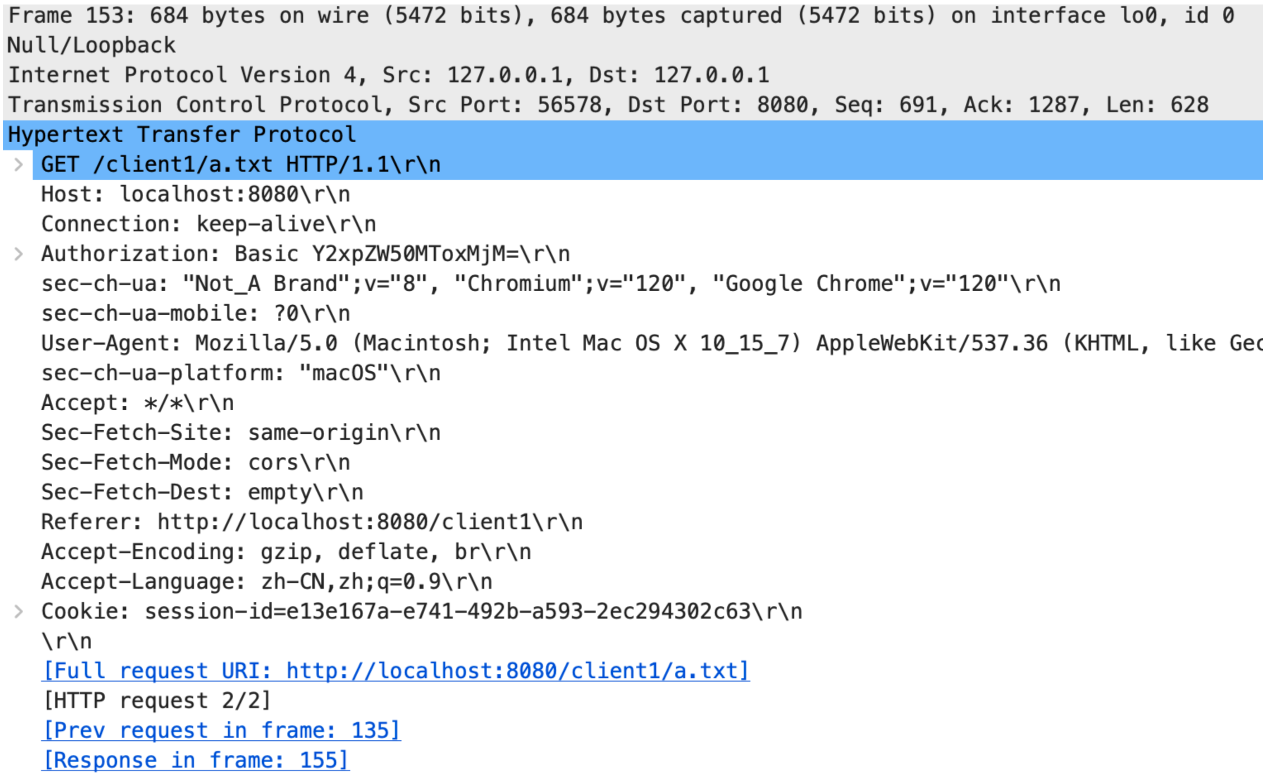
头部和消息主体示例

为了说明头部的使用和HTTP消息主体的内容，让我们通过Wireshark查看一些示例：

1. GET请求：

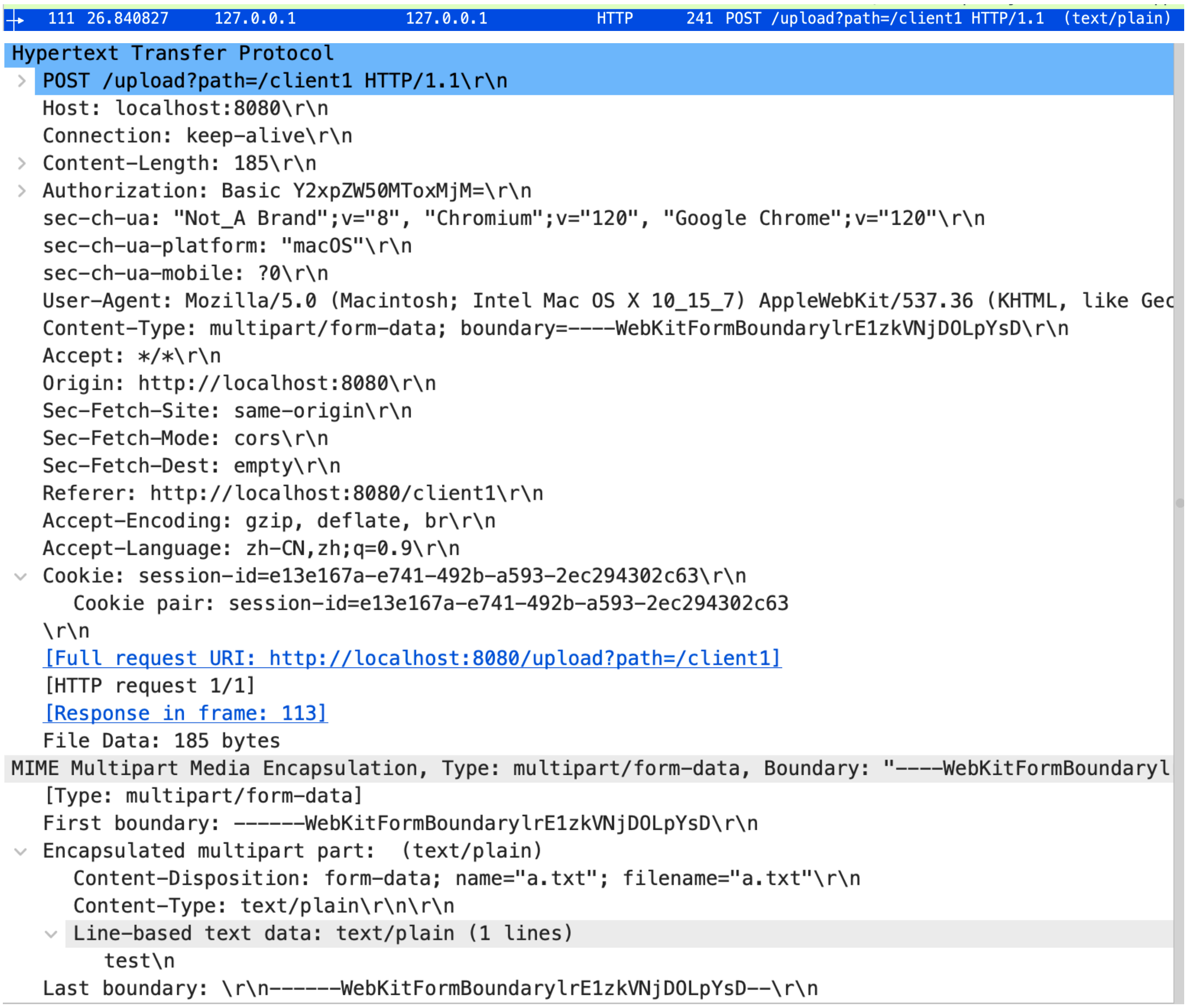


```
   153  30.394246     127.0.0.1            127.0.0.1            HTTP      684  GET /client1/a.txt HTTP/1.1
```



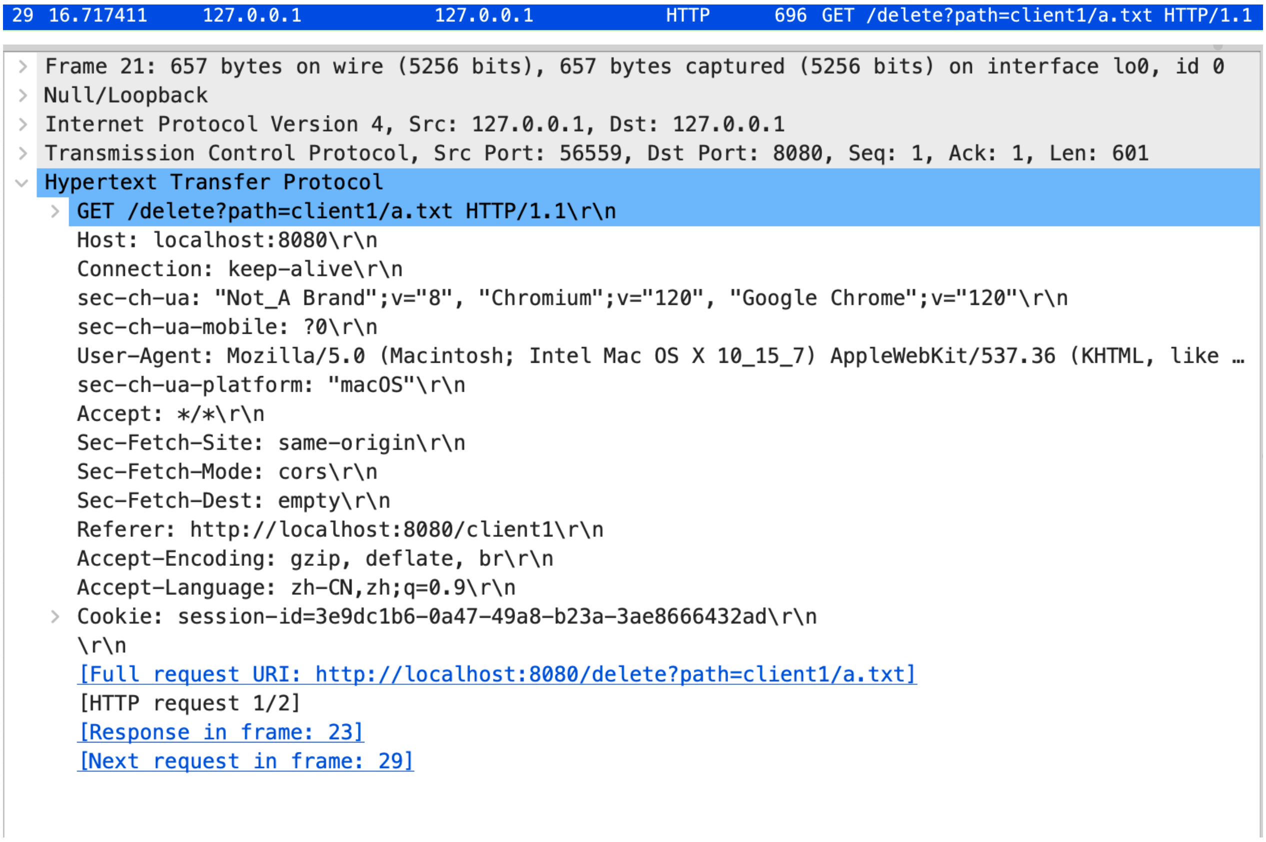
```
Frame 153: 684 bytes on wire (5472 bits), 684 bytes captured (5472 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 56578, Dst Port: 8080, Seq: 691, Ack: 1287, Len: 628
Hypertext Transfer Protocol
 > GET /client1/a.txt HTTP/1.1\r\n
   Host: localhost:8080\r\n
   Connection: keep-alive\r\n
 > Authorization: Basic Y2xpZW50MToxMjM=\r\n
   sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"\r\n
   sec-ch-ua-mobile: ?0\r\n
   User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gec
   sec-ch-ua-platform: "macOS"\r\n
   Accept: */*\r\n
   Sec-Fetch-Site: same-origin\r\n
   Sec-Fetch-Mode: cors\r\n
   Sec-Fetch-Dest: empty\r\n
   Referer: http://localhost:8080/client1\r\n
   Accept-Encoding: gzip, deflate, br\r\n
   Accept-Language: zh-CN,zh;q=0.9\r\n
 > Cookie: session-id=e13e167a-e741-492b-a593-2ec294302c63\r\n
   \r\n
   [Full request URI: http://localhost:8080/client1/a.txt]
   [HTTP request 2/2]
   [Prev request in frame: 135]
   [Response in frame: 155]
```

2. POST请求：



```
 111 26.840827    127.0.0.1         127.0.0.1         HTTP     241 POST /upload?path=/client1 HTTP/1.1  (text/plain)
Hypertext Transfer Protocol
 > POST /upload?path=/client1 HTTP/1.1\r\n
   Host: localhost:8080\r\n
   Connection: keep-alive\r\n
 > Content-Length: 185\r\n
 > Authorization: Basic Y2xpZW50MToxMjM=\r\n
   sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"\r\n
   sec-ch-ua-platform: "macOS"\r\n
   sec-ch-ua-mobile: ?0\r\n
   User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gec
   Content-Type: multipart/form-data; boundary=----WebKitFormBoundarylrE1zkVNjDOLpYsD\r\n
   Accept: */*\r\n
   Origin: http://localhost:8080\r\n
   Sec-Fetch-Site: same-origin\r\n
   Sec-Fetch-Mode: cors\r\n
   Sec-Fetch-Dest: empty\r\n
   Referer: http://localhost:8080/client1\r\n
   Accept-Encoding: gzip, deflate, br\r\n
   Accept-Language: zh-CN,zh;q=0.9\r\n
 v Cookie: session-id=e13e167a-e741-492b-a593-2ec294302c63\r\n
      Cookie pair: session-id=e13e167a-e741-492b-a593-2ec294302c63
   \r\n
   [Full request URI: http://localhost:8080/upload?path=/client1]
   [HTTP request 1/1]
   [Response in frame: 113]
   File Data: 185 bytes
MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "----WebKitFormBoundaryl
   [Type: multipart/form-data]
   First boundary: ------WebKitFormBoundarylrE1zkVNjDOLpYsD\r\n
 v Encapsulated multipart part:  (text/plain)
      Content-Disposition: form-data; name="a.txt"; filename="a.txt"\r\n
      Content-Type: text/plain\r\n\r\n
    v Line-based text data: text/plain (1 lines)
         test\n
   Last boundary: \r\n------WebKitFormBoundarylrE1zkVNjDOLpYsD--\r\n
```

3. DELETE请求：

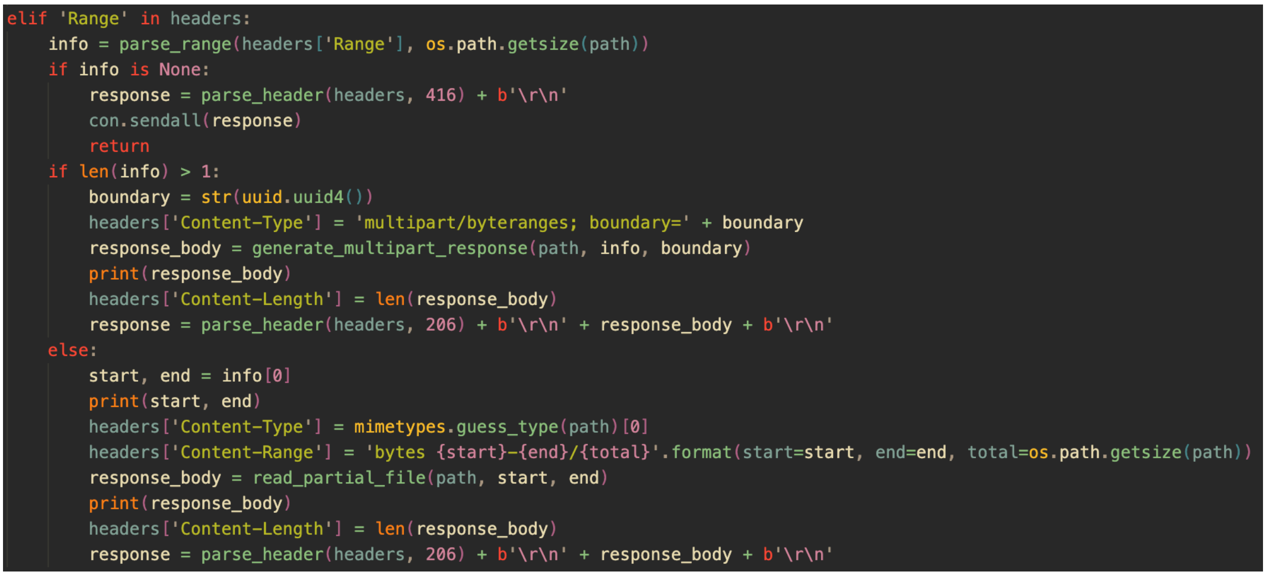


```
29 16.717411      127.0.0.1             127.0.0.1             HTTP       696  GET /delete?path=client1/a.txt HTTP/1.1
```

```
> Frame 21: 657 bytes on wire (5256 bits), 657 bytes captured (5256 bits) on interface lo0, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 56559, Dst Port: 8080, Seq: 1, Ack: 1, Len: 601
v Hypertext Transfer Protocol
  > GET /delete?path=client1/a.txt HTTP/1.1\r\n
    Host: localhost:8080\r\n
    Connection: keep-alive\r\n
    sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"\r\n
    sec-ch-ua-mobile: ?0\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like …
    sec-ch-ua-platform: "macOS"\r\n
    Accept: */*\r\n
    Sec-Fetch-Site: same-origin\r\n
    Sec-Fetch-Mode: cors\r\n
    Sec-Fetch-Dest: empty\r\n
    Referer: http://localhost:8080/client1\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Accept-Language: zh-CN,zh;q=0.9\r\n
  > Cookie: session-id=3e9dc1b6-0a47-49a8-b23a-3ae8666432ad\r\n
    \r\n
    [Full request URI: http://localhost:8080/delete?path=client1/a.txt]
    [HTTP request 1/2]
    [Response in frame: 23]
    [Next request in frame: 29]
```

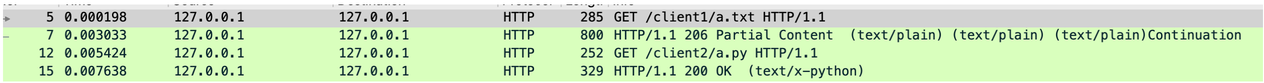
Bonus

Breakpoint Transmission

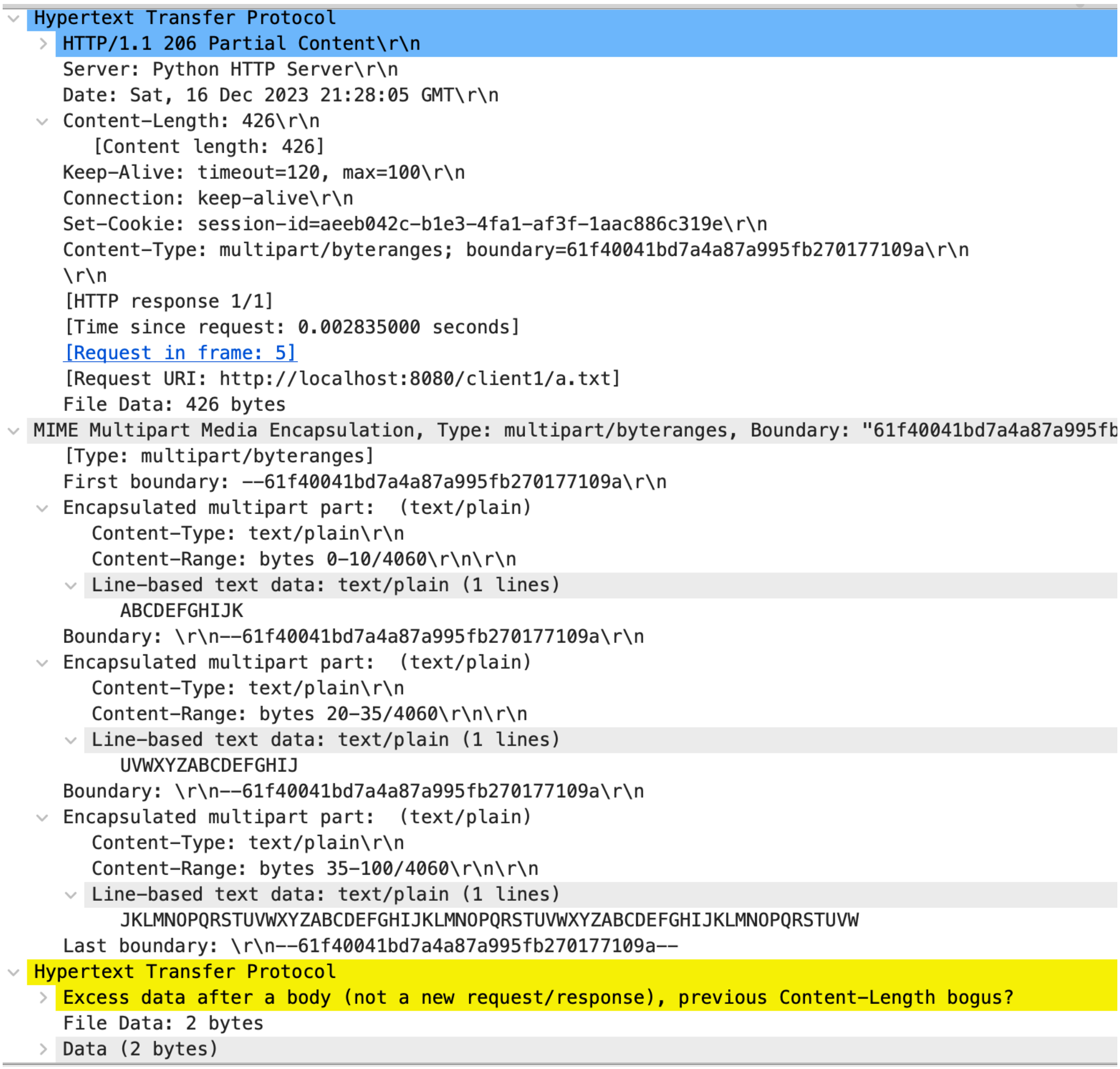


```python
elif 'Range' in headers:
    info = parse_range(headers['Range'], os.path.getsize(path))
    if info is None:
        response = parse_header(headers, 416) + b'\r\n'
        con.sendall(response)
        return
    if len(info) > 1:
        boundary = str(uuid.uuid4())
        headers['Content-Type'] = 'multipart/byteranges; boundary=' + boundary
        response_body = generate_multipart_response(path, info, boundary)
        print(response_body)
        headers['Content-Length'] = len(response_body)
        response = parse_header(headers, 206) + b'\r\n' + response_body + b'\r\n'
    else:
        start, end = info[0]
        print(start, end)
        headers['Content-Type'] = mimetypes.guess_type(path)[0]
        headers['Content-Range'] = 'bytes {start}-{end}/{total}'.format(start=start, end=end, total=os.path.getsize(path))
        response_body = read_partial_file(path, start, end)
        print(response_body)
        headers['Content-Length'] = len(response_body)
        response = parse_header(headers, 206) + b'\r\n' + response_body + b'\r\n'
```

单个文件和多个文件的切片传输需要分别处理，分别设置不同的Content-Type包括request header的内容

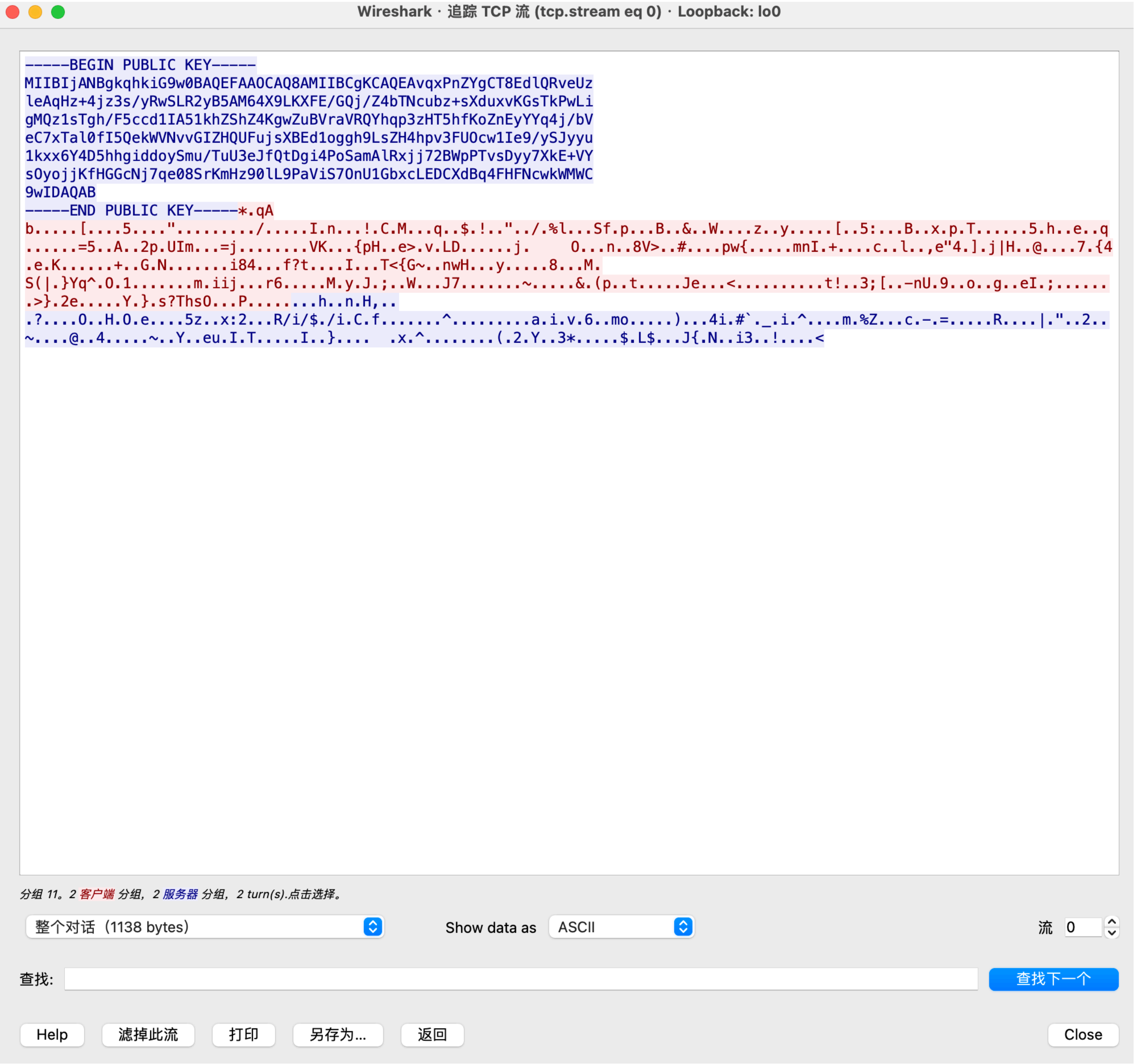


```
   5 0.000198    127.0.0.1         127.0.0.1         HTTP     285 GET /client1/a.txt HTTP/1.1
   7 0.003033    127.0.0.1         127.0.0.1         HTTP     800 HTTP/1.1 206 Partial Content  (text/plain) (text/plain) (text/plain)Continuation
  12 0.005424    127.0.0.1         127.0.0.1         HTTP     252 GET /client2/a.py HTTP/1.1
  15 0.007638    127.0.0.1         127.0.0.1         HTTP     329 HTTP/1.1 200 OK  (text/x-python)
```



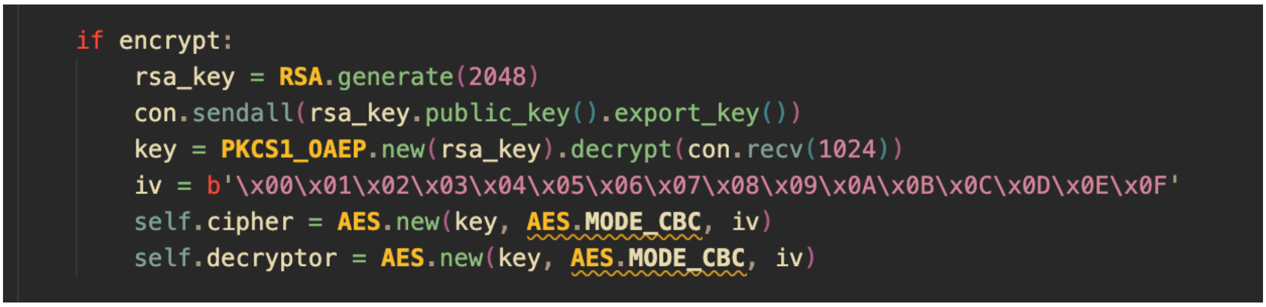
```
∨ Hypertext Transfer Protocol
  > HTTP/1.1 206 Partial Content\r\n
    Server: Python HTTP Server\r\n
    Date: Sat, 16 Dec 2023 21:28:05 GMT\r\n
  ∨ Content-Length: 426\r\n
       [Content length: 426]
    Keep-Alive: timeout=120, max=100\r\n
    Connection: keep-alive\r\n
    Set-Cookie: session-id=aeeb042c-b1e3-4fa1-af3f-1aac886c319e\r\n
    Content-Type: multipart/byteranges; boundary=61f40041bd7a4a87a995fb270177109a\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.002835000 seconds]
    [Request in frame: 5]
    [Request URI: http://localhost:8080/client1/a.txt]
    File Data: 426 bytes
∨ MIME Multipart Media Encapsulation, Type: multipart/byteranges, Boundary: "61f40041bd7a4a87a995fb
    [Type: multipart/byteranges]
    First boundary: --61f40041bd7a4a87a995fb270177109a\r\n
  ∨ Encapsulated multipart part:  (text/plain)
       Content-Type: text/plain\r\n
       Content-Range: bytes 0-10/4060\r\n\r\n
     ∨ Line-based text data: text/plain (1 lines)
          ABCDEFGHIJK
    Boundary: \r\n--61f40041bd7a4a87a995fb270177109a\r\n
  ∨ Encapsulated multipart part:  (text/plain)
       Content-Type: text/plain\r\n
       Content-Range: bytes 20-35/4060\r\n\r\n
     ∨ Line-based text data: text/plain (1 lines)
          UVWXYZABCDEFGHIJ
    Boundary: \r\n--61f40041bd7a4a87a995fb270177109a\r\n
  ∨ Encapsulated multipart part:  (text/plain)
       Content-Type: text/plain\r\n
       Content-Range: bytes 35-100/4060\r\n\r\n
     ∨ Line-based text data: text/plain (1 lines)
          JKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVW
    Last boundary: \r\n--61f40041bd7a4a87a995fb270177109a--
∨ Hypertext Transfer Protocol
  > Excess data after a body (not a new request/response), previous Content-Length bogus?
    File Data: 2 bytes
  > Data (2 bytes)
```

Encryption



```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvqxPnZYgCT8EdlQRveUz
leAqHz+4jz3s/yRwSLR2yB5AM64X9LKXFE/GQj/Z4bTNcubz+sXduxvKGsTkPwLi
gMQz1sTgh/F5ccd1IA51khZShZ4KgwZuBVraVRQYhqp3zHT5hfKoZnEyYYq4j/bV
eC7xTal0fI5QekWVNvvGIZHQUFujsXBEd1oggh9LsZH4hpv3FUOcw1Ie9/ySJyyu
1kxx6Y4D5hhgiddoySmu/TuU3eJfQtDgi4PoSamAlRxjj72BWpPTvsDyy7XkE+VY
sOyojjKfHGGcNj7qe08SrKmHz90lL9PaViS7OnU1GbxcLEDCXdBq4FHFNcwkWMWC
9wIDAQAB
-----END PUBLIC KEY-----*.qA
b.....[....5....".........../.....I.n...!.C.M...q..$.!.."../.%l...Sf.p...B..&..W....z..y.....[..5:...B..x.p.T......5.h..e..q
......=5..A..2p.UIm...=j.........VK...{pH..e>.v.LD......j.    O...n..8V>..#....pw{.....mnI.+....c..l..,e"4.].j|H..@....7.{4
.e.K......+..G.N.......i84...f?t....I...T<{G~..nwH...y.....8...M.
S(|.}Yq^.O.1........m.iij...r6.....M.y.J.;..W...J7.......~.....&.(p..t.....Je...<...........t!..3;[..-nU.9..o..g..eI.;......
.>}.2e.....Y.}.s?ThsO...P........h..n.H,..
.?....O..H.O.e....5z..x:2...R/i/$./i.C.f.......^.........a.i.v.6..mo.....)...4i.#`._.i.^....m.%Z...c.-.=.....R....|."..2..
~....@..4.....~..Y..eu.I.T.....I..}....  .x.^........(.2.Y..3*.....$.L$...J{.N..i3..!....<
```

分组 11。2 客户端 分组，2 服务器 分组，2 turn(s).点击选择。



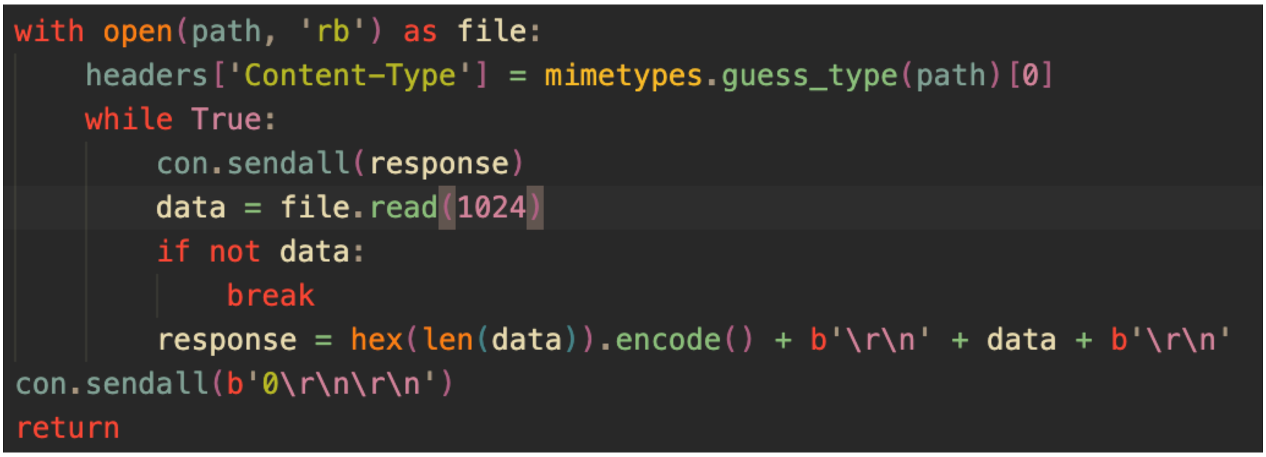
```python
if encrypt:
    rsa_key = RSA.generate(2048)
    con.sendall(rsa_key.public_key().export_key())
    key = PKCS1_OAEP.new(rsa_key).decrypt(con.recv(1024))
    iv = b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
    self.cipher = AES.new(key, AES.MODE_CBC, iv)
    self.decryptor = AES.new(key, AES.MODE_CBC, iv)
```

加密的具体流程为：

1. Client 发起连接，接收到Server端生成的RSA公钥
2. Client 生成AES密钥，并用接收到的公钥加密密钥
3. Client 发送密文给 Server端，Server用自己的私钥解密，得到`key`
4. 双方通过AES对称密钥通信，其中初始向量为硬编码的`iv`（约定俗成）

加密后的流量如上图所示，第一段明文`PEM`格式的RSA公钥，随后的数据传输皆为密文形式。

SpeedTest for File Downloading

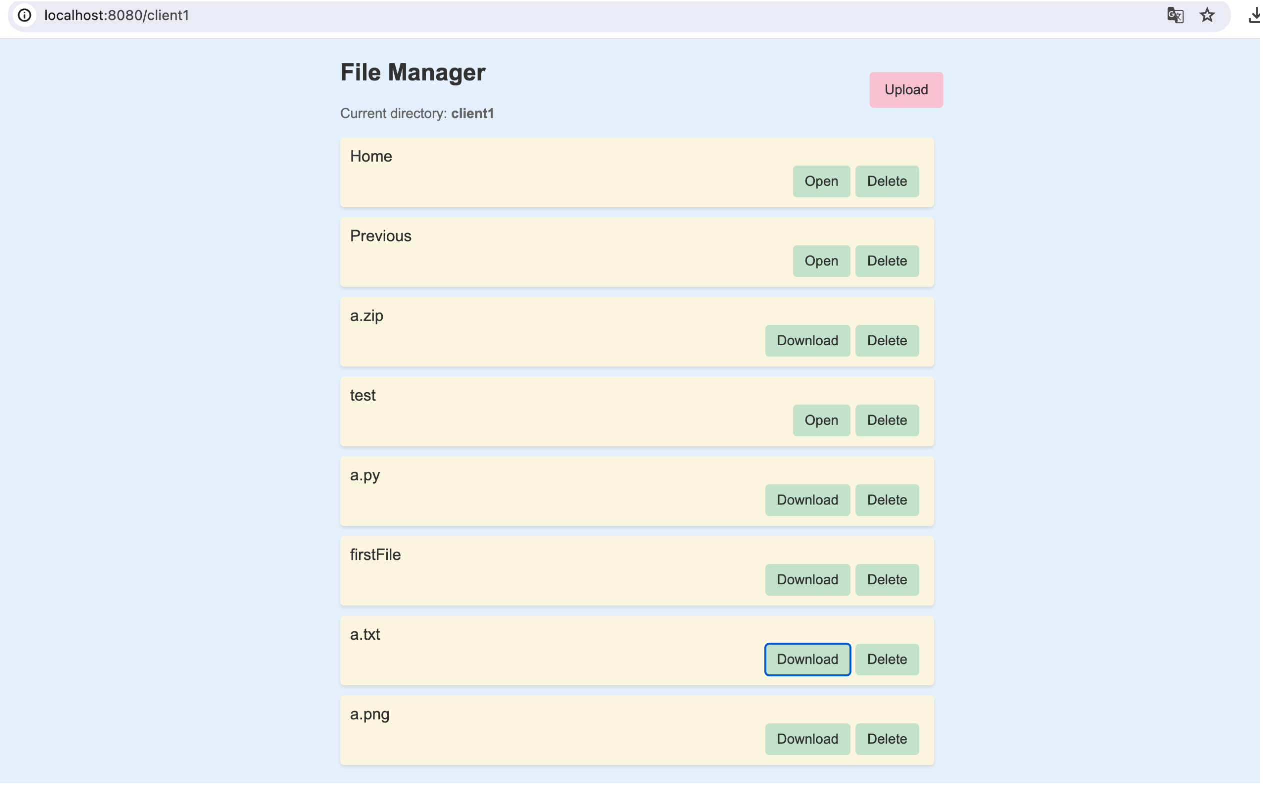


```
with open(path, 'rb') as file:
    headers['Content-Type'] = mimetypes.guess_type(path)[0]
    while True:
        con.sendall(response)
        data = file.read(1024)
        if not data:
            break
        response = hex(len(data)).encode() + b'\r\n' + data + b'\r\n'
con.sendall(b'0\r\n\r\n')
return
```

在传输大文件的时候，使用chunked Transfer，不需要等文件完全读取成功之后统一发送，而是以1024个字节为一个单位，封装后发送，节省了时间。

Other Bonus

美观简约的交互界面，可以支持上传下载删除包括最开始的认证，以及返回根目录、上级目录，访问文件夹和显示当前目录的功能



localhost:8080/client1

File Manager

Current directory: **client1**

Home

Previous

a.zip

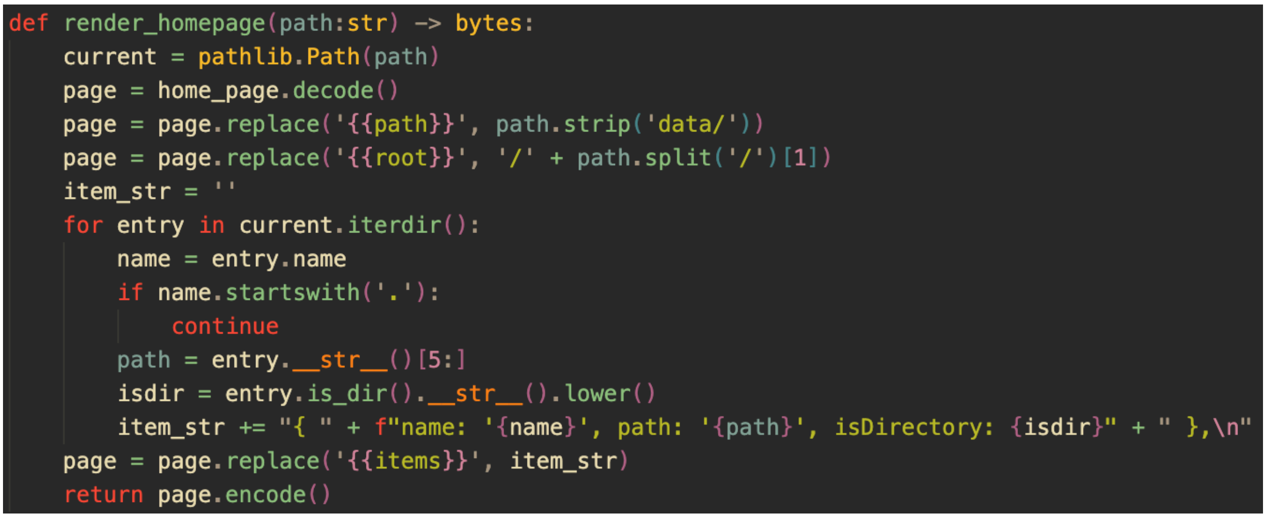
test

a.py

firstFile

a.txt

a.png



```python
def render_homepage(path:str) -> bytes:
    current = pathlib.Path(path)
    page = home_page.decode()
    page = page.replace('{{path}}', path.strip('data/'))
    page = page.replace('{{root}}', '/' + path.split('/')[1])
    item_str = ''
    for entry in current.iterdir():
        name = entry.name
        if name.startswith('.'):
            continue
        path = entry.__str__()[5:]
        isdir = entry.is_dir().__str__().lower()
        item_str += "{ " + f"name: '{name}', path: '{path}', isDirectory: {isdir}" + " },\n"
    page = page.replace('{{items}}', item_str)
    return page.encode()
```

以及相应的HTML文件(index.html)

完善的异常处理，包括信息的认证，命令的格式，以及上传下载文件的存在性检查等

结论

通过这个项目，我们成功地实现了一个基于HTTP/1.1协议的文件管理服务器。我们的服务器能够处理各种文件管理操作，并提供了目录列表、文件下载、文件上传和文件删除的功能。我们的自定义框架和数据结构使我们能够高效地处理和表达HTTP消息的各个部分。